

Model-based generation of natural language specifications

Phan Vo Thu Nhat, Maria Spichkova

RMIT University, Melbourne, Australia

s3220976@student.rmit.edu.au, maria.spichkova@rmit.edu.au

Abstract. Application of formal models provides many benefits for the software and system development, however, the learning curve of formal languages could be a critical factor for an industrial project. Thus, a natural language specification that reflects all the aspects of the formal model might help to understand the model and be especially useful for the stakeholders who do not know the corresponding formal language. Moreover, an *automated generation* of the documentation from the model would replace manual updates of the documentation for the cases the model is modified. This paper presents an ongoing work on generating natural language specifications from formal models. Our goal is to generate documentation in English from the basic modelling artefacts, such as data types, state machines, and architectural components. To allow further formal analysis of the generated specification, we restrict English to its subset, Attempto Controlled English.¹

1 Introduction

Model-based development (MBD) is a paradigm in which software and system development focus on high-level executable models, cf. [34]. In the early development phases, formal models allow a wide range of exploration and analysis using domain-specific notations in order to simplify the system design, development or verification/testing. Application of formal models provides many benefits for the software and system development. In “40 years of formal methods” [5], Bjørner and Havelund admit that the gap between academic research on formal methods and its integration in large industrial projects is yet to be bridged. There are a number of hindering factors for adoption of formal methods in industry [33]. As crucial obstacles can be named lack of understandability and readability [29,32], and our aim is to find appropriate ways to avoid these obstacles. Also, human factors play a crucial role and have to be taken into account [28,31].

Application of formal models requires an interplay between formal and informal methods, which use different levels of formality in descriptions. A manual solution to this problem was suggested many years ago: Guiho and Hennebert reported a communication problem in the SACEM project [15] between the verifiers and other engineers, who were not familiar with the formal specification

¹ Preprint. Accepted to the Software Technologies: Applications and Foundations (STAF 2016). Final version published by Springer International Publishing AG.

method. The problem was solved by providing the engineers with a natural language description derived *manually* from the formal specification. For a large-scale projects, it would be too time-consuming to derive a natural language specification (NLS) manually. In this paper, we propose a framework for *automated generation* of NLS from the basic modelling artefacts, such as data type definitions, State Transition Diagrams (STDs), and architecture specifications.

Contributions: The proposed solution would serve not only increasing the understandability of formal models, but also keeping the system documentation up-to-date. System documentation is an important part of the development process, but it is often considered by industry as a secondary appendage to the main part of the development – modelling and implementation. It is hard to keep the documentation up-to-date if the system model is frequently changing during the modelling phase of the development. Thus, system requirements documents and the general systems description are not updated according to the system’s or model’s modifications. Sometimes the updates are overlooked, sometimes they are omitted on purpose. For example, it is because of timing or costs constraints on the project. As a result, the system documentation is often outdated and does not describe the latest version of the system model. The question is whether we need to update the documentation *manually*, cf. [32].

Outline: The rest of the paper is organised as follows. Section 2 describes the related work. Section 3 introduces the proposed framework and a small case study to illustrate the ideas of the framework. In Section 4 we summarise the paper and propose directions for future research.

2 Related work

The research field of automated translation from formal modelling languages to natural languages is almost uncovered, however, there are many approaches on automated generation of (semi-)formal specifications from natural language ones. Lee and Bryant [23] presented an approach automatically generate formal specifications in an object-oriented notation from NLS. Cabral and Sampaio [9] suggested to use a Controlled Natural Language (CNL), a subset of English to analyse system characteristics represented by a set of declarative sentences. CNL use restricted vocabulary, grammar rules in defined knowledge based for the aim of formal models generation. This also allows to generate structured models at different levels of abstraction, as well as provides formal refinement of user actions and system responses.

Schwitter et al. [27] introduced ECOLE, an editor for a controlled language called PENG (Process-able English), that defines a mapping between English and First-Order Logic in order to verify requirements consistency, as well as to help writing manuals and system specifications to improve documentation quality, which is our goal of generated specifications in natural language.

As several attempts have been made to automate the requirement capture, there is another approach for the automatic construction of Object-oriented design model in UML diagram from natural language requirement specification.

Mala and Uma [24] present a methodology that utilizes the automatic reference resolution and eliminates the user intervention. The input problem statement is split into sentences for tagging by sentence splitter in order to get parts of speech for every word. The nouns and verbs are then identified by tagged texts based on simple phrasal grammars. Reference resolver is used to remove ambiguity by pronouns. The final text is then simplified by the normaliser for mapping the words into object-oriented system elements. The result produced by the system is compared with human output on the basic analysis of the text. The approach is promising to introduce a method to restructure the natural language text into modelling language in respect of system requirements specifications. Although there is a shortage of the efficiency in the tagger and reference resolver that result in unnatural expressions and misunderstandings, it can be improved by building a knowledge base for the system elements generation.

Juristo et al. [20] introduced an approach to formalise the requirement analysis process. The goal of this approach was to generate conceptual models in a precise manner, which provides support for resolving difficulties of misunderstanding the system requirements. The approach is based on examining the information extraction at the beginning of the development process (i.e., describing the problems in natural language sentences), and consists of two different activities: formalisation of the conceptual model and creation of the formal model. The limitation of this approach is in the difficulties to retrieve the rigorous and concise problem descriptions.

Gangopadhyay [14] suggested to design a conceptual model from a functional model, expressed in natural language sentences. Although its application is mainly for database applications, it can be extended to other design problems such as Web engineering and data warehousing. In order to interpret natural language expressions, Gangopadhyay applied the theory of Conceptual Dependencies developed by Schank, cf. [26]. The main goal of this approach was to identify data elements from functional model expressed in NLS, to locate missing information, as well as to integrate all individual data elements into an overall conceptual schema for data model establishment. A prototype system using Oracle database management system has been implemented to contain a parser for information collection. However, the lexicon in use is developed incrementally and semi-automated, so domain specialists still need to manually categorise words and phrases, to ensure non-relevant words are included in the system during the development of the conceptual model and to prevent systematic bias.

Bryant [8] suggested the theory of Two-Level Grammar for natural language requirements specification, in conjunction with Specification Development Environment to allow user interaction to refine model concepts. This approach allows the automation of the process of transition from requirements to design and implementation, as well as producing an understandable document on which software system will base on.

Ilieva and Ormandjieva [19] proposed an approach on transition of natural language software requirements specification into formal presentation. The au-

thors decided their method into three main processing parts: (1) the Linguistic Component as the text sentences to be analysed; (2) the Semantic Network as the formal NL presentation; and (3) modelling as the final phase of formal presentation of the specification. However, the approach of Ilieva and Ormandjieva involves manual human analysis process, to break down problems into smaller parts that are easily understood.

3 Framework

Figure 1 illustrates the general ideas of the suggested framework. To build a prototype for generation of NLS from the basic modelling artefacts, we have selected the AutoFocus3 modelling tool [4,16] as the basis for our models, because this tool (1) embeds the basic modelling artefacts, (2) is open source, as well as (3) has a well defined formal syntax behind all its modelling elements.

AutoFocus3 is developed on system models based on the FOCUS theory [7] that allows to specify system on different levels of abstraction formally and precisely. Source code of AutoFocus3 models are coded in XML, which makes it easy to parse and to analyse. AutoFocus3 has many advantages and is constantly evolving through last 10 years. The tool was applied as a part of tool chain within a number of development methodologies, e.g., for safety-critical systems in general [30,17,18], and for automotive-systems [11,10]. The tool can also be successfully applied for service-oriented modelling [6], which gives us another reason to select AutoFocus3 for the framework we develop.

To allow further formal analysis of the generated specification, we restrict English to its subset, Attempto Controlled English (ACE), cf. [13]. Specifications written in ACE give the impression of being informal, though they are in fact formal and machine executable. ACE provides a set of principles and recommendations for the strategy: to reduce the amount of lexical resources and structural sentences for a specification text to be unambiguously represented, and to fulfil the communication gap between domain specialist and software developer. Basically, the construct of ACE specification is the declarative sentence that is expressive enough to allow both natural usage and computer-processed purpose [12].

Implementation: We are currently implementing an automated translator from the AutoFocus3 models to ACE sentences in the Python programming language. Python was chosen as the development language due to its rapid prototyping features, as well as due to its increasing uptake by researchers as a scientific software development language because of good code readability and maintainability. With regard to the Python performance, it is sufficient for many common tasks and turns out to be very close to C language for parsing a file and a tree-like structure, cf. [25]. For the execution environment, we will research on the installation of ACE parsing engine, cf. [21], to execute natural language sentences in Prolog, cf. [3].

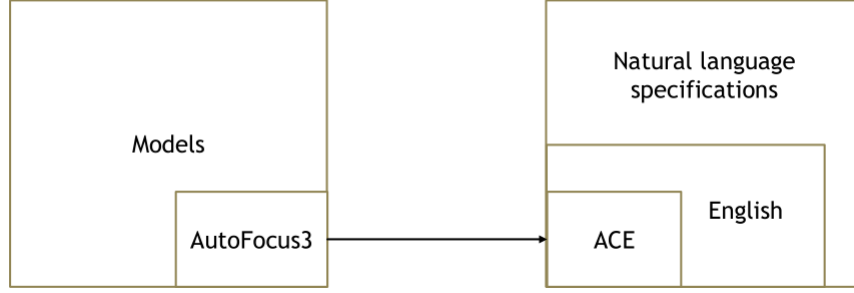


Fig. 1. Framework: Generation of natural language specifications from formal models

XML code of AutoFocus3 models. While parsing the XML code of an AutoFocus3 model, we have to identify three core sections:

- Specifications of data types and functions/constants (introduced by the XML-tag *rootElements* with the type *Data Dictionary*, cf. below for an example from the SimpleTrafficLight case study).
- Specifications of the system and components architecture (introduced by the XML-tag *rootElements* with the type *ComponentArchitecture*);
- Specifications of the state machines, used to describe the behaviour of system components (introduced by the XML-tag *containedElements* with the type *StateAutomaton*):

As each of these parts consists of XML representation of the AutoFocus3 elements, we can define a translation schema for each of these elements to generate English sentences out of the XML code. The sentences should conform to the ACE rules. To validate that this constraint is fulfilled, we have to analyse syntax and semantics of the generated sentences.

Translation schema. Let us discuss the translation schema in more details, focusing for simplicity on the specifications of data types and functions/constants. The definition of each data type is provided within the XML-tag *typeDefinitions*, where the keyword *Enumeration* indicates that this is an enumeration type. The name of the data type is coded within the attribute *name*. The elements of the type are introduced with the tag *members*. For the case of an enumeration type, we would have the following XML structure, where N is a natural number representing a number of elements in the data type, and i_1, \dots, i_{N+1} are some natural numbers representing internal identifiers of AutoFocus3 elements:

```

<typeDefinitions xsi:type="org-fortiss-af3-expression-definitions:Enumeration" id="i1"
name="TypeName">
  <members id="i2" name="MemberName1" />
  ...
  <members id="iN+1" name="MemberNameN" />

```

</typeDefinitions>

To generate an ACE sentence from this structure, we define two templates:

- For the case we have only one element, i.e., $N = 1$, we would use the template
 TypeName is a datatype. It consists-of one element that is MemberName_1 .
- For the case we have more than one element, i.e., $N > 1$, we would use the template
 TypeName is a datatype. It consists-of N elements that are $\text{MemberName}_1, \dots, \text{MemberName}_N$.

The definition of each function/constant is provided within the tag *function*, where its name and value are coded within the attributes *name* and *value*. For the case of constant function, we would have the following XML structure, where j_1, j_2 are some natural numbers representing internal identifiers of AutoFocus3 elements:

```
<functions id="j1" >
<function id="j2" name="ConstantName" />
<definition>
<statements xsi:type="org-fortiss-af3-expression-terms-imperative:Return" >
<value xsi:type="org-fortiss-af3-expression-terms:IntConst" value="ConstantVaue" />
</statements>
</definition>
<returnType xsi:type="org-fortiss-af3-expression-types:TInt" />
</functions>
```

To generate an ACE sentence from this structure, we define the following template:

ConstantName is a constant. It is equal to ConstantVaue .

Similar translation patterns apply for architecture specifications and state transition diagram sections.

ACE: Syntax check. ACE supports declarative sentences, which includes simple sentences, there is/are-sentences, boolean formulas, composite sentences, interrogative sentences, imperative sentences. ACE construction rules determine whether an English sentence is an ACE sentence, cf. [1]. Each ACE sentence is an acceptable English sentence, but not every English sentence is justified as a valid ACE sentence. Thus, to be conformed to ACE construction rules, an NLS in English should be constructed from the following elements:

- Function words: determiners, quantifiers, coordinators, negation words, pronouns, query words, modal auxiliaries, “be”, Saxon genitive marker’s;
- Fixed phrases: “there is”, “it is true that”;
- Content words: nouns, verbs, adjectives, adverbs, prepositions.

The function words and fixed phrases are predefined and cannot be changed, whereas content words can be modified by users within the lexicon format, cf. [2]. The content words cannot contain blank spaces. For instance, “interested in” should be reformulated to “interested-in”.

ACE: Semantics check. The mentioned above rules cannot remove all ambiguities in English. To avoid ambiguity, ACE provides a set of interpretation rules. Thus, each ACE sentence can have only one meaning, based on its syntax and on syntax of previous sentences.

The correctness of the generated sentences can be validated by the ACE query sentences, cf. [12]. They can be subdivided into three forms that are *yes/no*-questions (questions that require answer “yes” or “no”), *wh*-questions (questions starting with the words “What”, “When”, “Where”, etc.), and *how much/many*-questions, cf. [1]. For example, we could use the following questions to check the definition of an enumeration data type *XDataType*:

- What is *XDataType*?
- How many elements does *XDataType* have?
- Is *SomeElementName* an element of *XDataType*?

Case study: SimpleTrafficLight system. We present the core ideas of the framework on example of a small case study, Simple Traffic Lights, introduced by Lam and Teufl in [22]. In the Simple Traffic Lights case study, we the following elements in the data definitions section:

- Functions *tGreen*, *tRed*, and *tYellow* that return a constant integer value to represent the time in seconds for the active pedestrian or traffic light.
- Enumeration data types:
 - *pedestrianColor*: pedestrian lights (*Stop*, *Walk*);
 - *TrafficColor*: traffic lights (*Green*, *Red*, *RedYellow*, *Yellow*);
 - *Signal*: one-element data type to represent the *Present* signal;
 - *IndicatorSignal*: pedestrian requests to pass the street (*Off*, *On*).

Figure 2 illustrates the translation process from the AutoFocus3 data types and the corresponding XML descriptions, to ACE sentences. After translation, we check the definition of each data type as shown on Table 1 and in Figure 3.

In a similar manner the natural language description of the system and components architecture as well as of state machines, representing components behaviour, are generated and checked.

Table 1. Validation the generated sentences using ACE-questions

| Question | Answer |
|--|--------------------|
| What is IndicatorSignal? | It is a data-type. |
| How many elements does IndicatorSignal have? | It has 4 elements. |
| Is On an element of IndicatorSignal? | Yes, it is. |

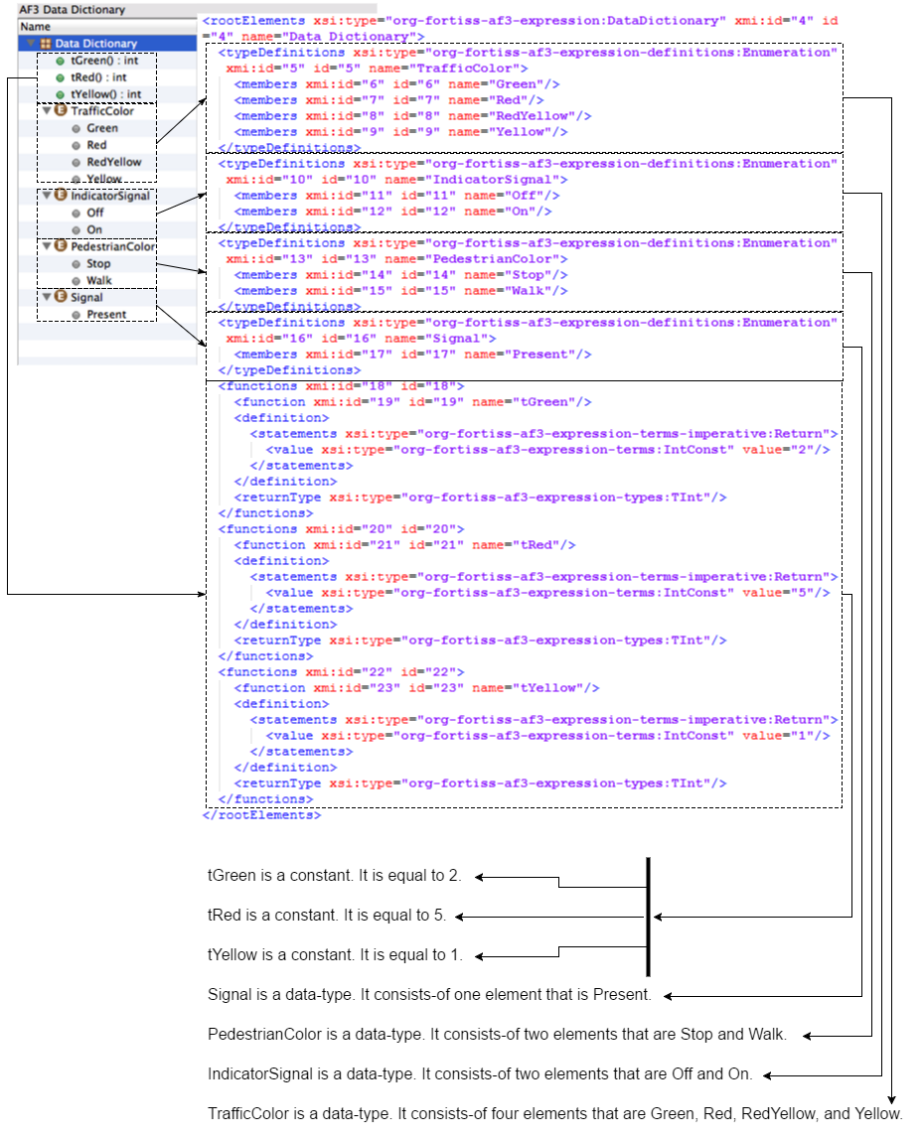


Fig. 2. Mapping from AutoFocus 3 data types to ACE sentences

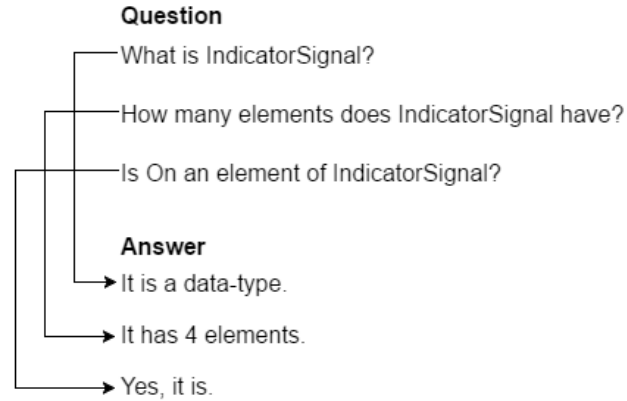


Fig. 3. Validation the generated sentences using ACE-questions

4 Conclusions and Future Work

This paper introduces our ongoing work on NLS from formal models. The goal of our current work is to generate documentation in English from the basic modelling artefacts of the AutoFocus3 modelling language, that are data types, state machines, and architectural components. This would allow to have an easy-to-read and easy-to-understand specifications of systems-under-development, written in English. To allow further formal analysis of the generated specification, we restrict English to its subset, ACE. The proposed framework, in its current version, can be applied to build a prototype for generation of ACE specifications from the AutoFocus3 models.

The future work focuses on the implementation of an prototype translator from AutoFocus3 to ACE, as well as on the extension of the framework to other formal modelling languages.

References

1. ACE Construction Rules. Accessed online 28-July-2016
http://attempto.ifi.uzh.ch/site/docs/ace_constructionrules.html.
2. ACE Lexicon Specification. Accessed online 28-July-2016
http://attempto.ifi.uzh.ch/site/docs/ace_lexicon.html.
3. SWI-Prolog. Accessed online 28-July-2016 at
<http://www.swi-prolog.org>.

4. V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. *Joint proceedings of ACES-MB 2015-Model-based Architecting of Cyber-physical and Embedded Systems*, page 19, 2015.
5. D. Bjørner and K. Havelund. 40 years of formal methods. In *FM 2014: Formal Methods*, pages 42–61. Springer, 2014.
6. M. Broy, J. Fox, F. Hölzl, D. Koss, M. Kuhrmann, M. Meisinger, B. Penzenstadler, S. Rittmann, B. Schätz, M. Spichkova, et al. Service-oriented modeling of cocome with focus and autofocus. In *The Common Component Modeling Example*, pages 177–206. Springer Berlin Heidelberg, 2008.
7. M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.
8. B. R. Bryant. Object-oriented natural language requirements specification. In *23rd Australasian Computer Science Conference*, pages 24–30. IEEE, 2000.
9. G. Cabral and A. Sampaio. Formal specification generation from requirement documents. *Electronic Notes in Theoretical Computer Science*, 195:171–188, 2008.
10. M. Feilkas, A. Fleischmann, F. Hölzl, C. Pfaller, K. Scheidemann, M. Spichkova, and D. Trachtenherz. A top-down methodology for the development of automotive software. Technical Report TUM-I0902, TU München, 2009.
11. M. Feilkas, F. Hölzl, C. Pfaller, S. Rittmann, B. Schätz, W. Schwitzer, W. Sitou, M. Spichkova, and D. Trachtenherz. A refined top-down methodology for the development of automotive software systems - the keylessentry system case study. Technical Report TUM-I1103, TU München, 2011.
12. N. E. Fuchs, K. Kaljurand, and T. Kuhn. Attempto Controlled English for knowledge representation. In *Reasoning Web*, pages 104–124. Springer, 2008.
13. N. E. Fuchs and R. Schwitter. Attempto Controlled English (ACE). *arXiv preprint cmp-lg/9603003*, 1996.
14. A. Gangopadhyay. Conceptual modeling from natural language functional specifications. *Artificial Intelligence in Engineering*, 15(2):207–218, 2001.
15. G. Guiho and C. Hennebert. Sacem software validation. In *12th International Conference on Software Engineering*, pages 186–191. IEEE, 1990.
16. F. Hölzl and M. Feilkas. AutoFocus 3 – A Scientific Tool Prototype for Model-Based Development of Component-Based, Reactive, Distributed Systems. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 317–322. Springer, 2010.
17. F. Hölzl, M. Spichkova, and D. Trachtenherz. AutoFocus Tool Chain. Technical Report TUM-I1021, TU München, 2010.
18. F. Hölzl, M. Spichkova, and D. Trachtenherz. Safety-critical system development methodology. Technical Report TUM-I1020, TU München, 2010.
19. M. Ilieva and O. Ormandjieva. Automatic transition of natural language software requirements specification into formal presentation. In *Natural Language Processing and Information Systems*, pages 392–397. Springer, 2005.
20. N. Juristo, J. L. Morant, and A. M. Moreno. A formal approach for generating OO specifications from natural language. *Journal of Systems and Software*, 48(2):139–153, 1999.
21. K. Kaljurand, N. E. Fuchs, and T. Kuhn. APE - ACE Parsing Engine. Online at <https://github.com/Attempto/APE> Accessed 30-March-2016.
22. P. S. Lam and S. Teufl. Simple Traffic Lights tutorial for AutoFocus 3. Online at <http://af3.fortiss.org/docs/> Accessed 30-March-2016.
23. B. Lee and B. R. Bryant. Automated conversion from requirements documentation to an object-oriented formal specification language. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 932–936. ACM, 2002.

24. G. A. Mala and G. Uma. Automatic construction of object oriented design models [UML diagrams] from natural language requirements specification. In *PRICAI 2006: Trends in Artificial Intelligence*, pages 1155–1159. Springer, 2006.
25. M. F. Sanner. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61, 1999.
26. R. C. Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive psychology*, 3(4):552–631, 1972.
27. R. Schwitter, A. Ljungberg, and D. Hood. ECOLE – A Look-ahead Editor for a Controlled Language. *EAMT-CLAW03*, pages 141–150, 2003.
28. M. Spichkova. Human Factors of Formal Methods. In *In IADIS Interfaces and Human Computer Interaction 2012*. IHCI 2012, 2012.
29. M. Spichkova. Design of formal languages and interfaces: “formal” does not mean “unreadable”. In K. Blashki and P. Isaias, editors, *Emerging Research and Trends in Interactivity and the Human-Computer Interface*. IGI Global, 2013.
30. M. Spichkova, F. Hölzl, and D. Trachtenherz. Verified System Development with the AutoFocus Tool Chain. In *Workshop on Formal Methods in the Development of Software*, 2012.
31. M. Spichkova, H. Liu, M. Laali, and H. W. Schmidt. Human factors in software reliability engineering. *Workshop on Applications of Human Error Research to Improve Software Engineering (WAHESE2015)*, 2015.
32. M. Spichkova, X. Zhu, and D. Mou. Do we really need to write documentation for a system? In *International Conference on Model-Driven Engineering and Software Development (MODELSWARD13)*, 2013.
33. A. Zamansky, G. Rodriguez-Navas, M. Adams, and M. Spichkova. Formal methods in collaborative projects. In *11th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. IEEE, 2016.
34. J. Zhang and B. H. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th international conference on Software engineering*, pages 371–380. ACM, 2006.